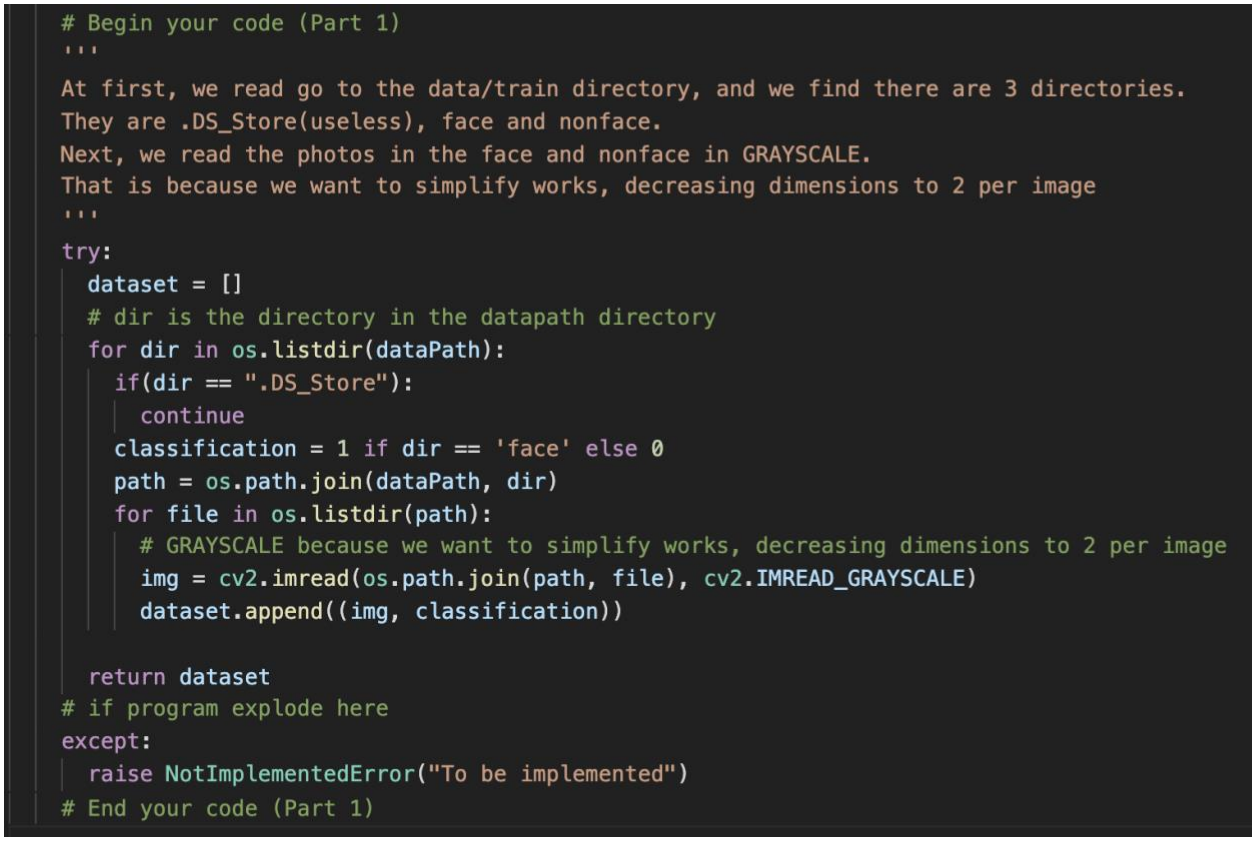
109550050

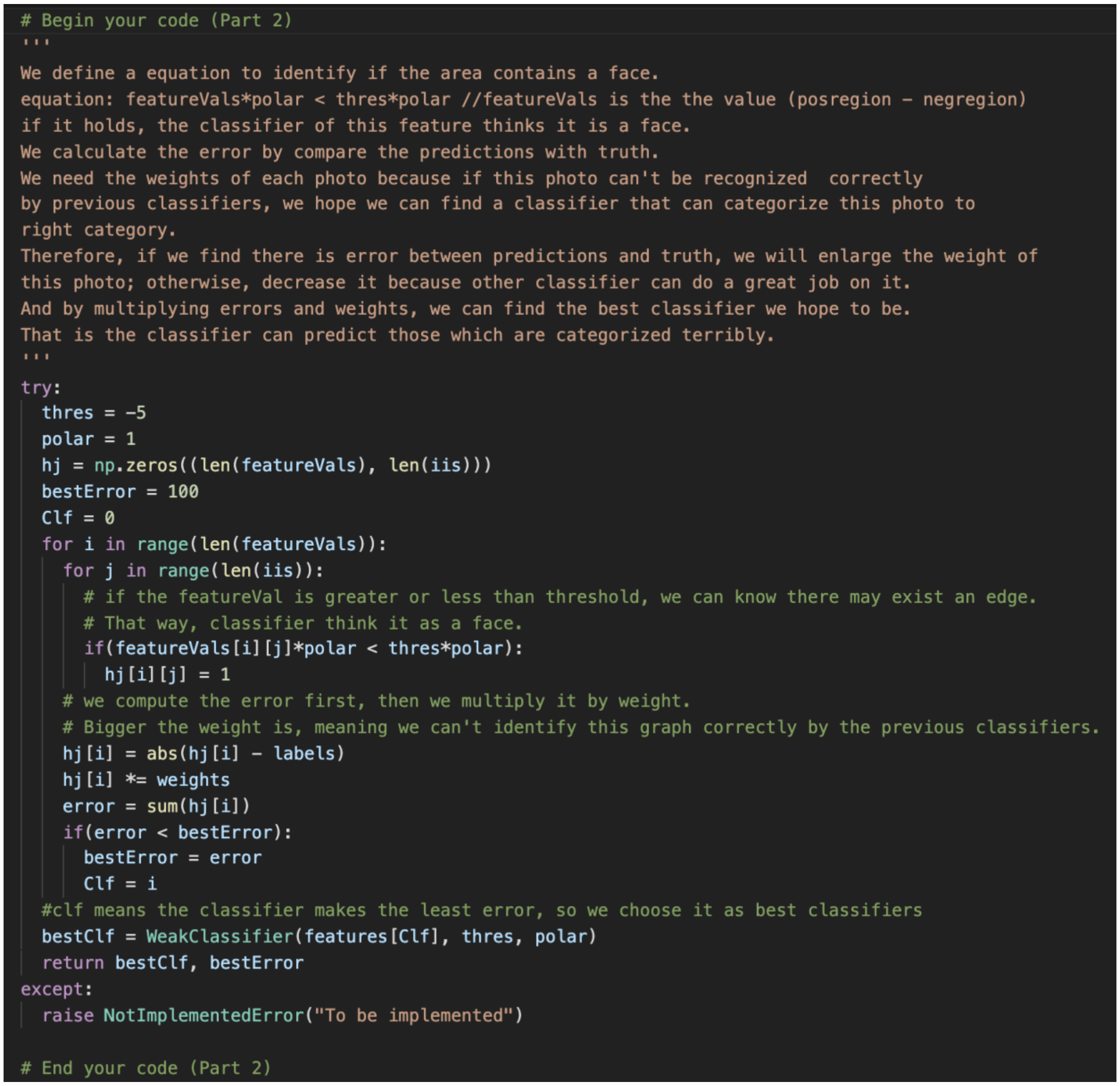
Part I. Implementation (6%):

Part1:



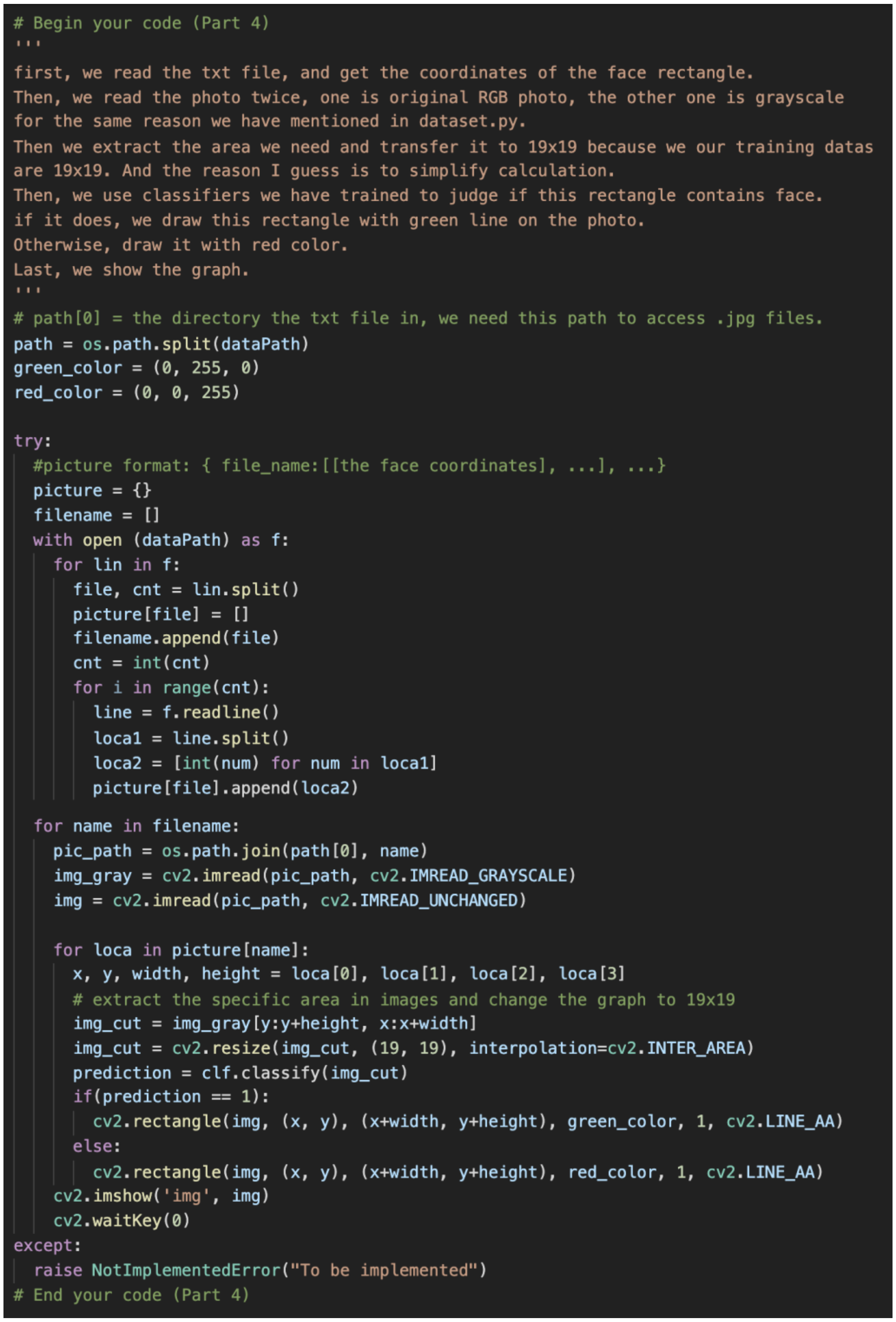
```
# Begin your code (Part 1)
'''
At first, we read go to the data/train directory, and we find there are 3 directories.
They are .DS_Store(useless), face and nonface.
Next, we read the photos in the face and nonface in GRAYSCALE.
That is because we want to simplify works, decreasing dimensions to 2 per image
'''
try:
    dataset = []
    # dir is the directory in the datapath directory
    for dir in os.listdir(dataPath):
        if(dir == ".DS_Store"):
            continue
        classification = 1 if dir == 'face' else 0
        path = os.path.join(dataPath, dir)
        for file in os.listdir(path):
            # GRAYSCALE because we want to simplify works, decreasing dimensions to 2 per image
            img = cv2.imread(os.path.join(path, file), cv2.IMREAD_GRAYSCALE)
            dataset.append((img, classification))

    return dataset
# if program explode here
except:
    raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

Part2:



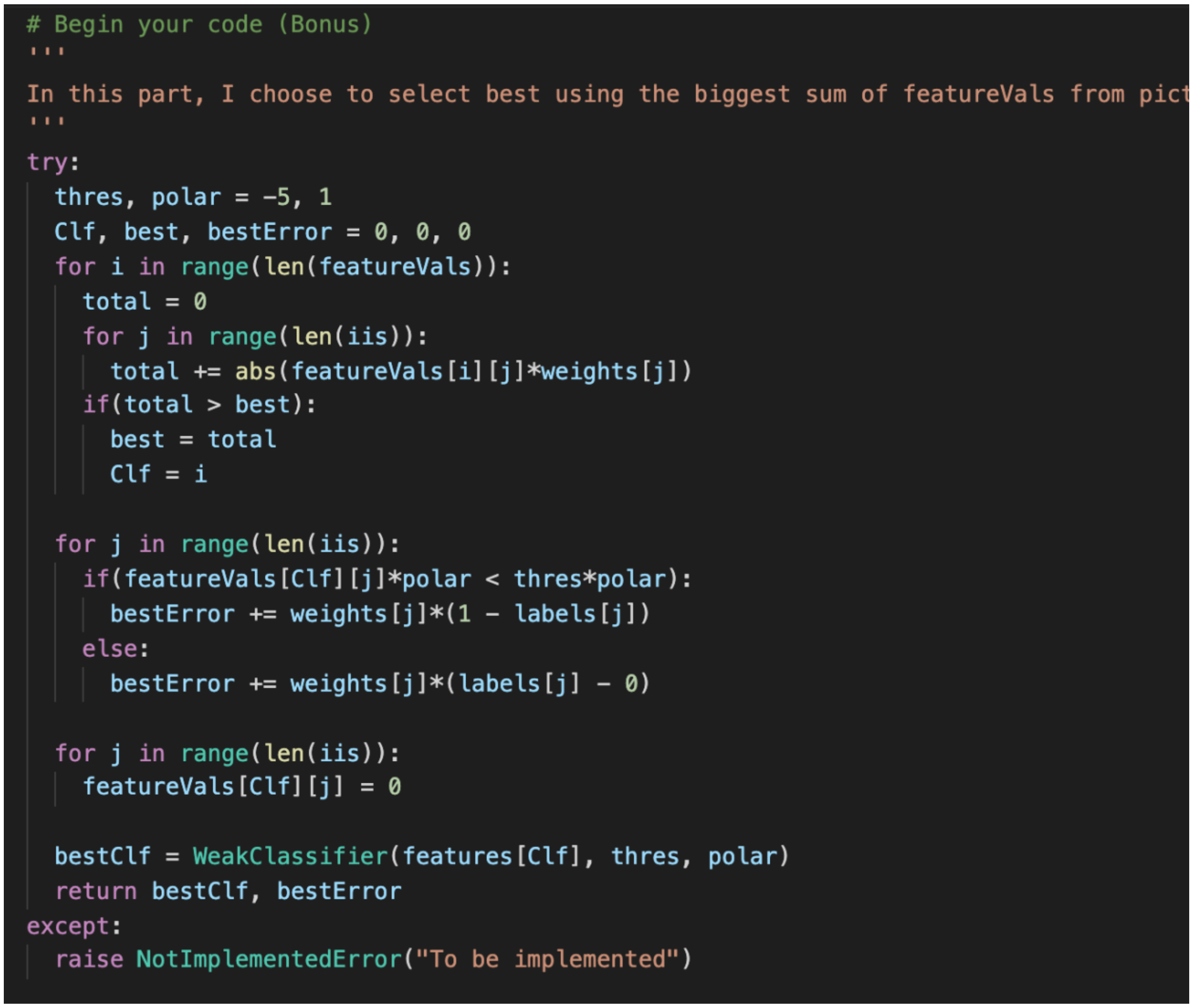
```
# Begin your code (Part 2)
'''
We define a equation to identify if the area contains a face.
equation: featureVals*polar < thres*polar //featureVals is the the value (posregion - negregion)
if it holds, the classifier of this feature thinks it is a face.
We calculate the error by compare the predictions with truth.
We need the weights of each photo because if this photo can't be recognized  correctly
by previous classifiers, we hope we can find a classifier that can categorize this photo to
right category.
Therefore, if we find there is error between predictions and truth, we will enlarge the weight of
this photo; otherwise, decrease it because other classifier can do a great job on it.
And by multiplying errors and weights, we can find the best classifier we hope to be.
That is the classifier can predict those which are categorized terribly.
'''
try:
  thres = -5
  polar = 1
  hj = np.zeros((len(featureVals), len(iis)))
  bestError = 100
  Clf = 0
  for i in range(len(featureVals)):
    for j in range(len(iis)):
      # if the featureVal is greater or less than threshold, we can know there may exist an edge.
      # That way, classifier think it as a face.
      if(featureVals[i][j]*polar < thres*polar):
        hj[i][j] = 1
    # we compute the error first, then we multiply it by weight.
    # Bigger the weight is, meaning we can't identify this graph correctly by the previous classifiers.
    hj[i] = abs(hj[i] - labels)
    hj[i] *= weights
    error = sum(hj[i])
    if(error < bestError):
      bestError = error
      Clf = i
  #clf means the classifier makes the least error, so we choose it as best classifiers
  bestClf = WeakClassifier(features[Clf], thres, polar)
  return bestClf, bestError
except:
  raise NotImplementedError("To be implemented")

# End your code (Part 2)
```

Part4:



```python
# Begin your code (Part 4)
'''
first, we read the txt file, and get the coordinates of the face rectangle.
Then, we read the photo twice, one is original RGB photo, the other one is grayscale
for the same reason we have mentioned in dataset.py.
Then we extract the area we need and transfer it to 19x19 because we our training datas
are 19x19. And the reason I guess is to simplify calculation.
Then, we use classifiers we have trained to judge if this rectangle contains face.
if it does, we draw this rectangle with green line on the photo.
Otherwise, draw it with red color.
Last, we show the graph.
'''
# path[0] = the directory the txt file in, we need this path to access .jpg files.
path = os.path.split(dataPath)
green_color = (0, 255, 0)
red_color = (0, 0, 255)

try:
  #picture format: { file_name:[[the face coordinates], ...], ...}
  picture = {}
  filename = []
  with open (dataPath) as f:
    for lin in f:
      file, cnt = lin.split()
      picture[file] = []
      filename.append(file)
      cnt = int(cnt)
      for i in range(cnt):
        line = f.readline()
        loca1 = line.split()
        loca2 = [int(num) for num in loca1]
        picture[file].append(loca2)

  for name in filename:
    pic_path = os.path.join(path[0], name)
    img_gray = cv2.imread(pic_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.imread(pic_path, cv2.IMREAD_UNCHANGED)

    for loca in picture[name]:
      x, y, width, height = loca[0], loca[1], loca[2], loca[3]
      # extract the specific area in images and change the graph to 19x19
      img_cut = img_gray[y:y+height, x:x+width]
      img_cut = cv2.resize(img_cut, (19, 19), interpolation=cv2.INTER_AREA)
      prediction = clf.classify(img_cut)
      if(prediction == 1):
        cv2.rectangle(img, (x, y), (x+width, y+height), green_color, 1, cv2.LINE_AA)
      else:
        cv2.rectangle(img, (x, y), (x+width, y+height), red_color, 1, cv2.LINE_AA)
    cv2.imshow('img', img)
    cv2.waitKey(0)
except:
  raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

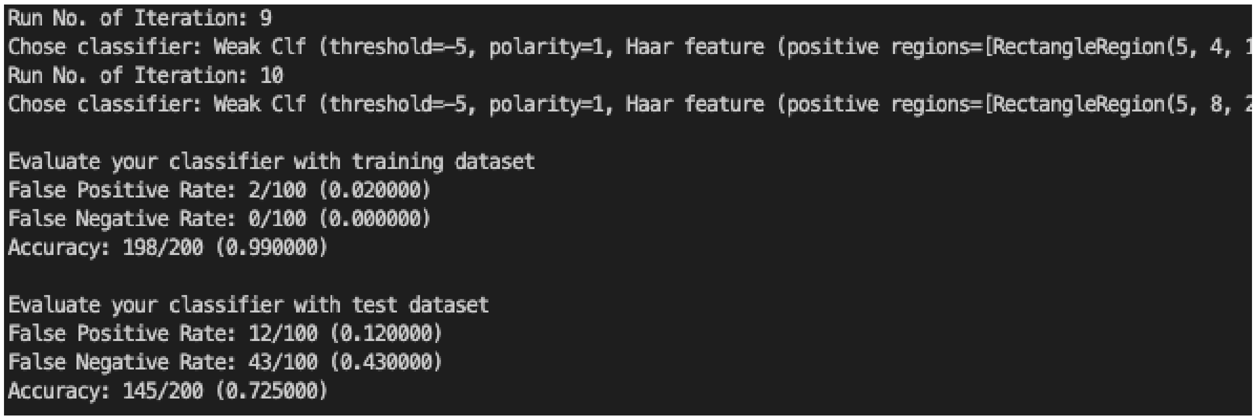
Bonus:



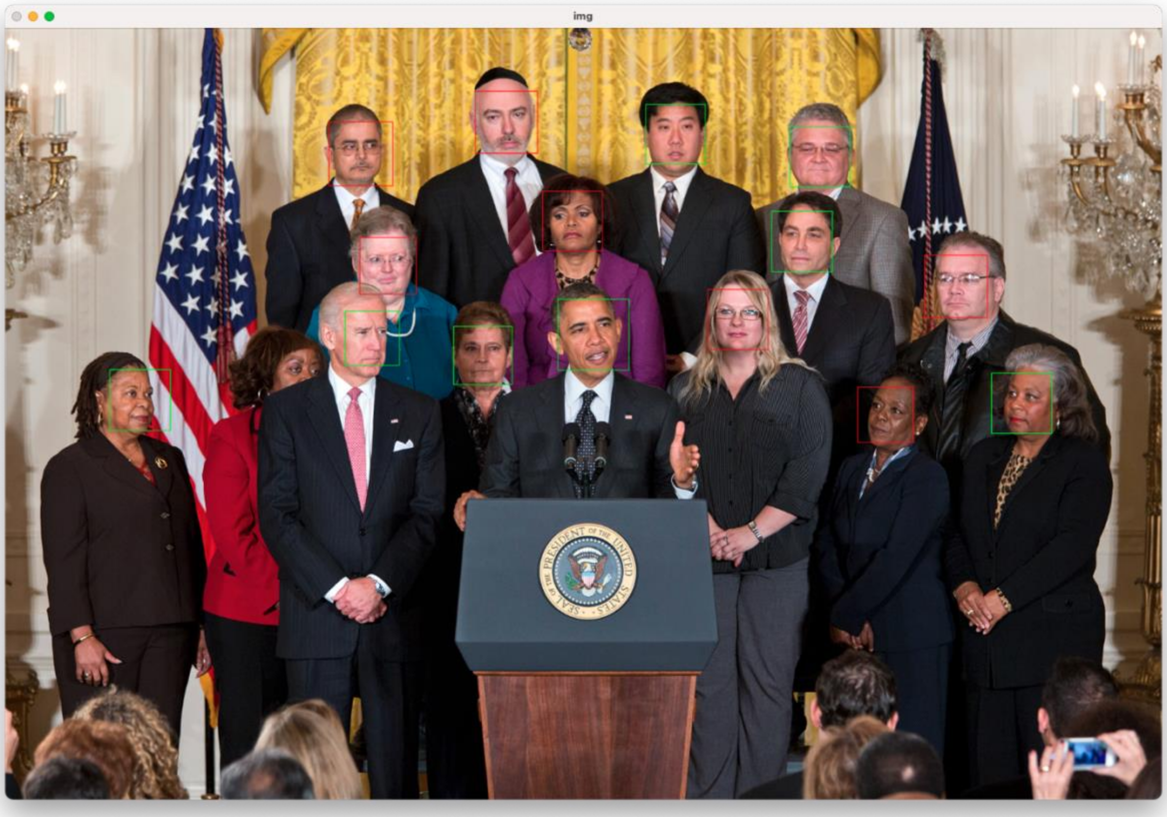
```
# Begin your code (Bonus)
'''
In this part, I choose to select best using the biggest sum of featureVals from pict
'''
try:
    thres, polar = -5, 1
    Clf, best, bestError = 0, 0, 0
    for i in range(len(featureVals)):
        total = 0
        for j in range(len(iis)):
            total += abs(featureVals[i][j]*weights[j])
        if(total > best):
            best = total
            Clf = i

    for j in range(len(iis)):
        if(featureVals[Clf][j]*polar < thres*polar):
            bestError += weights[j]*(1 - labels[j])
        else:
            bestError += weights[j]*(labels[j] - 0)

    for j in range(len(iis)):
        featureVals[Clf][j] = 0

    bestClf = WeakClassifier(features[Clf], thres, polar)
    return bestClf, bestError
except:
    raise NotImplementedError("To be implemented")
```

Part II. Results & Analysis (12%):

Results:

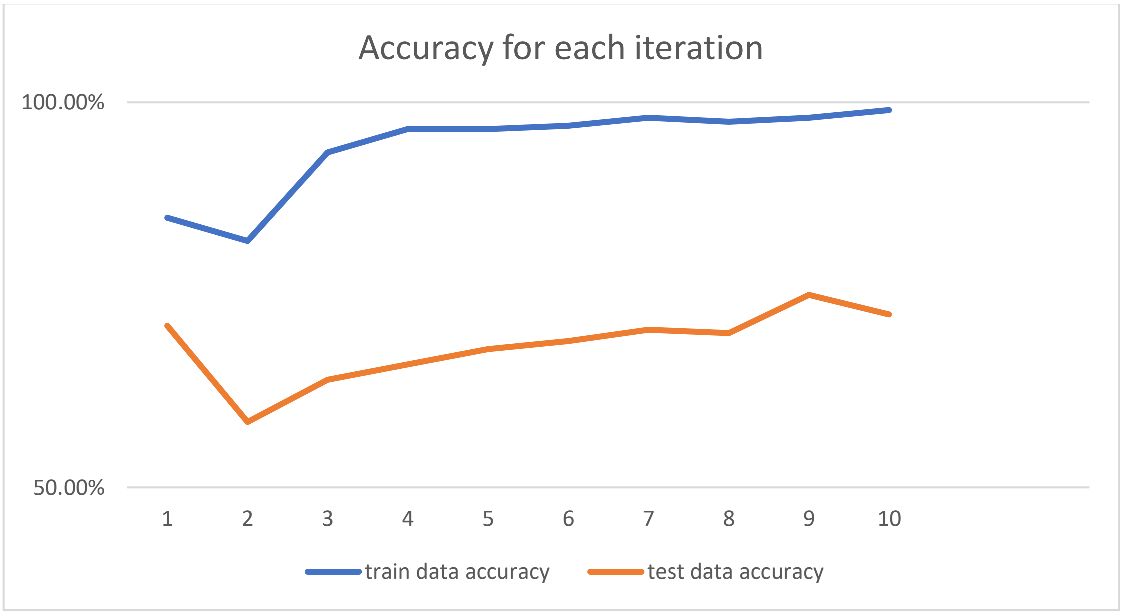


```
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=-5, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 4, 1
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=-5, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 8, 2

Evaluate your classifier with training dataset
False Positive Rate: 2/100 (0.020000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 198/200 (0.990000)

Evaluate your classifier with test dataset
False Positive Rate: 12/100 (0.120000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 145/200 (0.725000)
```

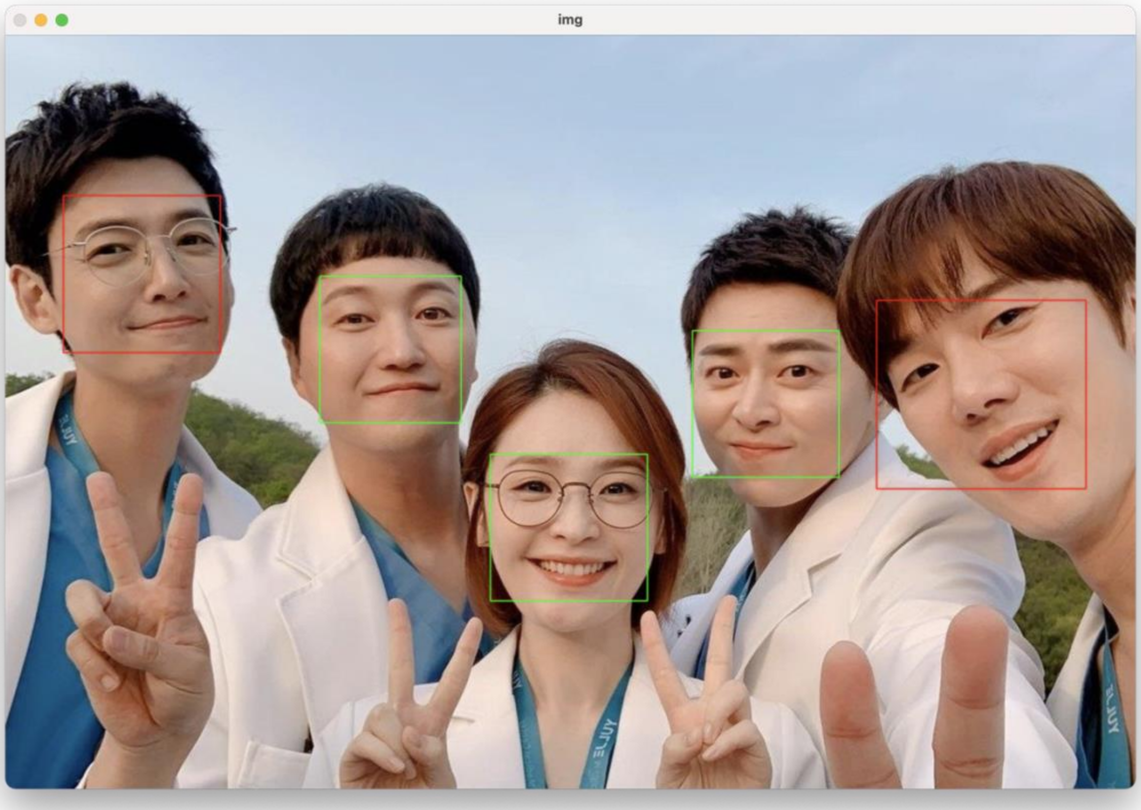


200 張	train data accuracy	test data accuracy
Method 1 t = 1	85.0%	71.0%
Method 1 t = 2	82.0%	58.5%
Method 1 t = 3	93.5%	64.0%
Method 1 t = 4	96.5%	66.0%
Method 1 t = 5	96.5%	68.0%
Method 1 t = 6	97.0%	69.0%
Method 1 t = 7	98.0%	70.5%
Method 1 t = 8	97.5%	70.0%
Method 1 t = 9	98.0%	75.0%
Method 1 t = 10	99.0%	72.5%

(table for t from 1 to 10)



(chart for t from 1 to 10)



(test using my own images with the strong classifier with t = 9)

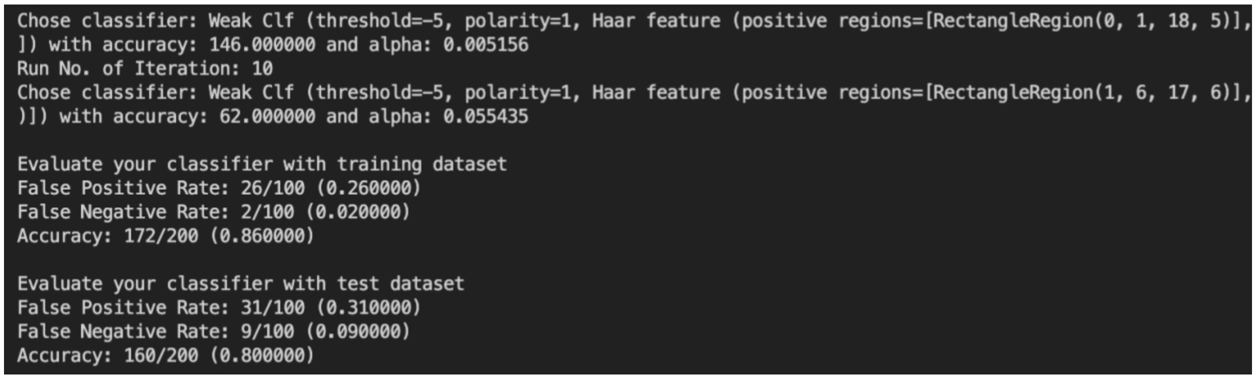
Analysis:

First, we can observe that classifiers have better performance on training data. This makes sense because we just use these data to train our classifiers. During training procedure, we always choose the feature that makes the least error with weights. Therefore, it is expected to perform well on training data. As for test data, because the classifiers have never been trained to identify them, there is lots of different face with unique features. It's reasonable that the classifiers make more errors on them than on those face they have been trained to identify, just as human.

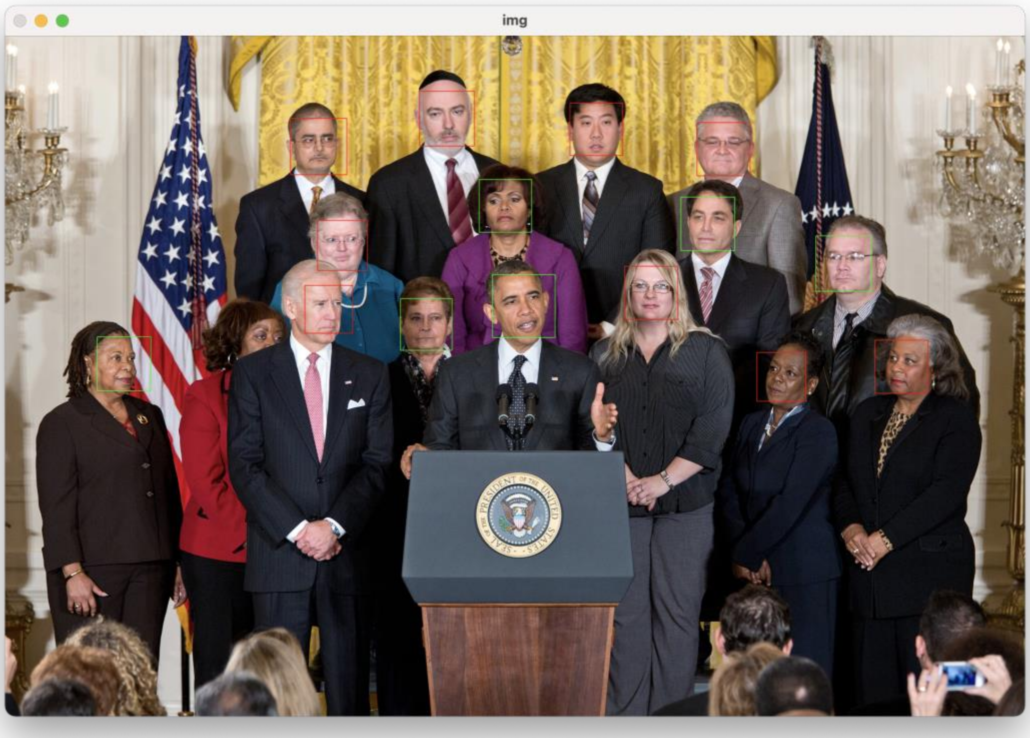
Let's discuss on t value. The value t means we train t weak classifiers and combine them to be a strong classifier. If t = 5, it means our strong classifier is a combination of five weak classifiers. In other words, we use five features in this strong classifier. From t = 1 to t = 10, we observe that t has positive correlation with accuracy no matter on training data or testing data. This is what we expect because more features always bring more details about this feature. One nonface picture may share common features with a face picture accidentally. However, we can eliminate this coincidence by choosing more features, meaning larger value of t. As we mentioned, features "ALWAYS" bring more details, this is true when we discuss on training data. When we choose more and more features, it means our classifier will get more and more fit on our training data. Therefore, we can see that the training line is smoothy. The line rises gradually, but some exception we will discuss later. As for testing data, we can see a little floating on it. We compare two cases, t = 9 and t = 10. We find that the case t = 9 is more precise than the case t = 10 on this situation. This is opposite to the training case, and this is also contrary to our common sense. The following reasons are just my guess for this just on one case. I have no confidence that it works on every situation, or maybe this is not the actual reason. In my opinion, when we let t = 10, our accuracy on training data is a very high number, 99 percent. I guess this cause the classifier we train is too fit to the training data to identify general case correctly. It's so-called overfitting although it's my guess. Besides, we can find there is an apparent decline on t = 2, and this happens on both training and testing data. This can't be overfitting again, because both of their accuracies are quite low. I think this happens when t is small. On this situation, every weak classifier has more power to affect the outcome. In this case, suppose there are two weak classifiers A and B, maybe A can do a great job on some part, and so can B. However, when we mix them, the outcome may be terrible. But when t getting larger, the strong classifier will become more stable because no one weak classifier can affect the outcome hugely.

Lastly, I want to mention what I find in my testing of my own images. Take the above picture with doctors as example. When we train a classifier, it's quite important for where the feature is. If someone doesn't face the camera or his head is tilt, it may make difficulty for a classifier to identify correctly, such as the rightmost one or leftmost one in the doctor picture. To improve this, we may need to train for a tilt face or other situations, or choose a feature that isn't influenced hugely by this problem. However, this depends on some luck.

Bonus:



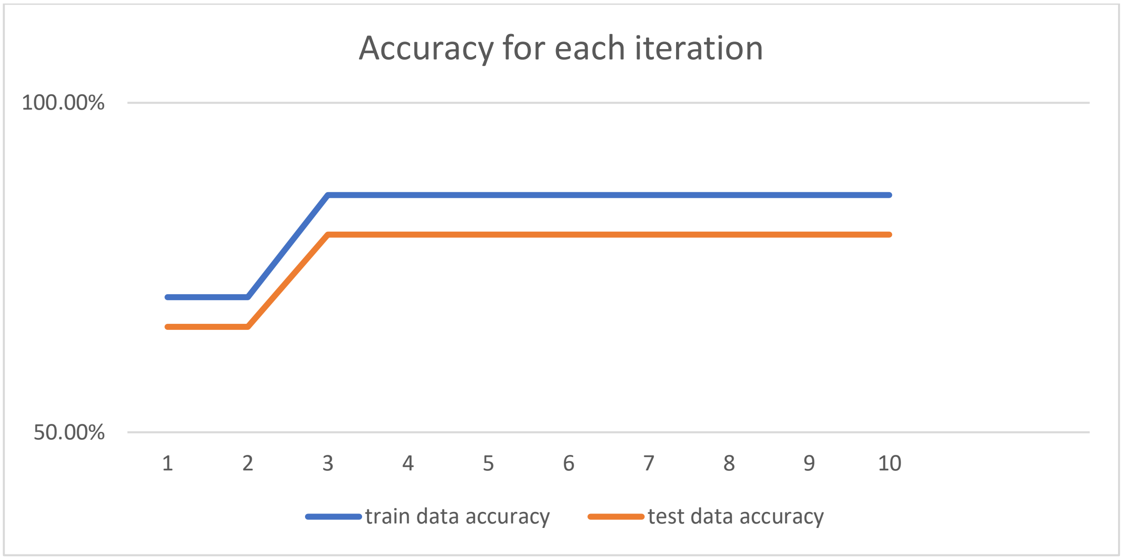
```
Chose classifier: Weak Clf (threshold=-5, polarity=1, Haar feature (positive regions=[RectangleRegion(0, 1, 18, 5)],
]) with accuracy: 146.000000 and alpha: 0.005156
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=-5, polarity=1, Haar feature (positive regions=[RectangleRegion(1, 6, 17, 6)],
)]) with accuracy: 62.000000 and alpha: 0.055435

Evaluate your classifier with training dataset
False Positive Rate: 26/100 (0.260000)
False Negative Rate: 2/100 (0.020000)
Accuracy: 172/200 (0.860000)

Evaluate your classifier with test dataset
False Positive Rate: 31/100 (0.310000)
False Negative Rate: 9/100 (0.090000)
Accuracy: 160/200 (0.800000)
```



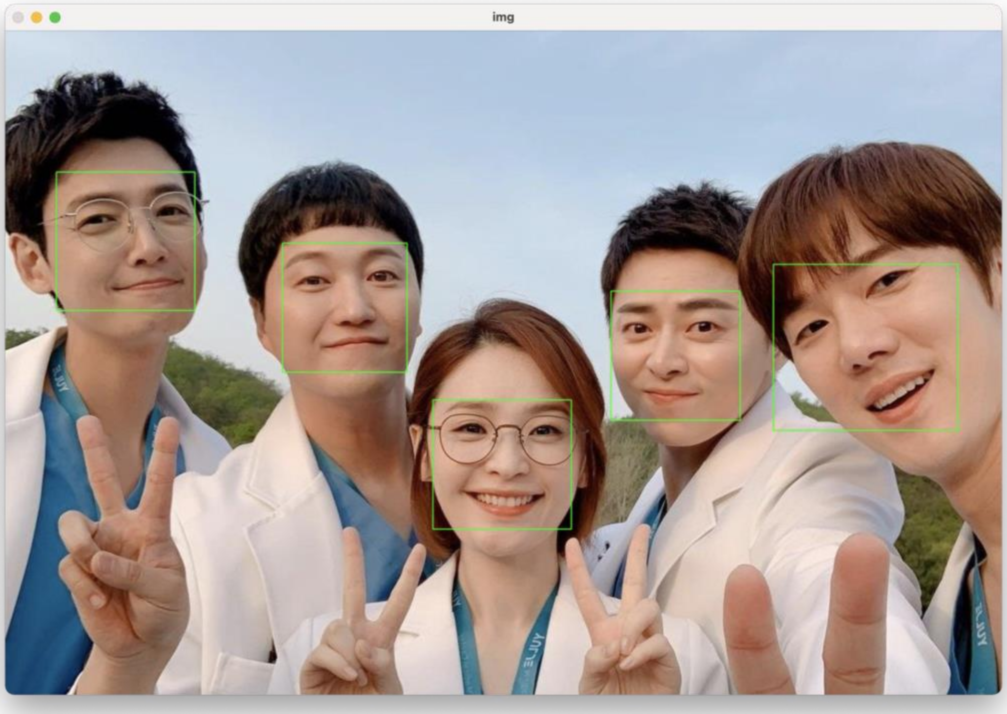
200 張	train data accuracy	test data accuracy
Method 1 t = 1	70.5%	66.0%
Method 1 t = 2	70.5%	66.0%
Method 1 t = 3	86.0%	80.0%
Method 1 t = 4	86.0%	80.0%
Method 1 t = 5	86.0%	80.0%
Method 1 t = 6	86.0%	80.0%
Method 1 t = 7	86.0%	80.0%
Method 1 t = 8	86.0%	80.0%

Method 1 t = 9	86.0%	80.0%
Method 1 t = 10	86.0%	80.0%

(table for t from 1 to 10)



(chart for t from 1 to 10)



(test using my own images with the strong classifier with t = 10)

Analysis:

In bonus part, I change the method we use to select best. I don’t use error rate as a standard; otherwise, I choose the biggest feature value as the best one. I think bigger the value is, more apparent the edge is. Hence, we can get a very distinct feature, and that must do a gorgeous job of course. However, there are lots of problems I encounter after implementing my idea. We can observe that with different t values, our accuracies don’t change a lot. To speak straightforwardly, there are just two different accuracies from t = 1 to 10. I will summarize the problems I find in the following lines. First, the value of feature can be impact greatly by the area of the feature region. If the region is very huge, then it’s value will greater in most situations. I have tried to overcome this by dividing area, but things went even worse. I guess there still some reasons I don’t come up with, at least now, I can’t deal with it. Second, weights have little to say in the select best weak classifiers. Due to the different methods to select and classify, the weights changing can’t really affect a lot for the following selecting. If the values of a feature are mostly contributed by the pictures of false prediction, we will make thing worse. To deal with it, I decrease the values of best feature to 0, preventing from selecting the same weak classifier. However, this doesn’t totally improve the problem that weights with no powers. Without influential weights, we can’t really learn from the previous weak classifiers. That is, the next selected weak classifier won’t zoom in the part that previous classifiers can’t recognize correctly, but we just still select the biggest one. This makes we always choose the features in the same region. These weak classifiers have lots of features in common. This is what makes accuracies are only two different values. In other words, we can say that the strong classifier does as much as there is still two weak classifiers in it when t = 10.

Apart from the disappointing part, there still something that is worthy of celebration. Luckily, we seem to choose the features that perform well on both training data and testing data. Its accuracy can get up to 80 percent, better than the original method. (Although I think this is all thanks to luck) What more, this can correctly recognize five doctors in my own image. Though this classifier (t = 10) tends to recognize everything as face (False positive rate is higher than the other one), I think it’s still ok. After all, it’s my own method.

Part III. Answer the questions (12%):

1. Please describe a problem you encountered and how you solved it.

   I think the most difficult period during doing this homework is at the beginning. Although I learned a little python when I was a freshman, I am still not very familiar with it. At first, I see lots of python documentation on the Net, such as numpy, cv2, etc. After overcoming the language problem, I suffer from the Viola-Jones'. Despite learning during class, I still confuse about some concept in Viola-Jones' and how can I implement it in python. To solve this problem, I find related information on the Net (Net is great, indeed), and spend lots of time reading TA's code. Gradually, I have a better understanding about the whole homework and Viola-Jones'.

2. What are the limitations of the **Viola-Jones' algorithm**?

   First of all, it costs lots of time on training classifiers. If we want to train a very strong classifier that can apply on most general situations, we may need to train it with huge amount of data, which costing lots of time. Second, we take face as example. If we train the classifier using only the photo of front face, it may do bad job with a turned face. Therefore, it seems not very flexible on some situations. The last I consider is it's sensitive to some factors, such as brightness.

3. Based on **Viola-Jones' algorithm**, how to improve the accuracy except increasing the training dataset and changing the parameter T?

   To improve accuracy, the first method is to increase image resolution. Higher the image resolution is, more detail we can find in the photos. That is, we can find more features, which may contain better features we don't have in low resolution. However, there is a side-affect with it – huge computation. Second, to get a more precise picture of face. As we known, the features in Viola-Jones' are very sensitive to the features' location. If we give a picture with a face at the right-bottom part of it, then we may get what we expect. Hence, get a precise picture is also an important part in Viola-Jones' algorithm, and it's not a part of the algorithm. We should improve it by any ways. The last but may be hard to make it is to change the method to choose features. During doing the bonus part of homework, I believe this can enhance the algorithm's performance. Although there is no one can come up with a better method to choose (otherwise, we may not use this method now), someday maybe someone will find it.

4. Please propose another possible **face detection** method (no matter how good or bad, please come up with an idea). Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.

   My idea is we select a set of pictures of face first. Then we compute all the haar-features of each picture. We select some features that all the pictures have use them to construct weak classifiers. If the testing picture has the feature value in an acceptable error, weak classifier recognizes it as face. Then as Viola-Jones', we combine all the weak classifiers to a strong classifier with some weights.

   The pros of this method are MAYBE we can easily find quite great features just by finding a large value of feature that every picture doesn't differ much. If this is available, I believe the method can have better performance than setting a threshold with a fixed value to classify. Of course, these pros just a beautiful imaginary in my mind. However, the cons are almost sure. First, we can't modify our error by changing weight in the adaboost algorithm. Once the feature really performs badly, we can do nothing but decrease its power on the strong classifier. Second, how to set a threshold for differing? If we choose a threshold that is too big to make even non-face can easily pass, then it can't do a great job. In the other hand, we will exclude lots of face if we choose too small one. The last, during the procedure, we need to use average value as a reference of a feature. There may be a doubt if the average can cause to lose its correctness.